Exhibit 11

Exhibit 11 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example Chase Count IV Systems and Services
U.S. Patent No. 7,712,080 ("the '080 Patent")

The Accused Systems and Services include, without limitation systems that utilize Apache Spark ("Spark"); all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Chase's systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example Chase Count IV Systems and Services").

U.S. Patent No. 7,712,080	
Example Claim 9	**Example Chase Count IV Systems and Services**
9. A distributed parallel computing system, having at least one memory area and at least one processor, comprising:	Upon information and belief, Chase's systems include "[a] distributed parallel computing system, having at least one memory area and at least one processor[.]" Upon information and belief, Chase's systems consist of a distributed parallel computing system having many memory areas and multiple processors. Upon information and belief, Chase's distributed parallel computing system is implemented, at least in part, using Spark. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. *See* https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 9, 2023).

9[a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area; and	Upon information and belief, Chase job postings confirm that Chase implements Spark in its data processing framework. **Job Responsibilities** • Design and train production grade ML models on large-scale datasets to solve various business use cases for Commercial Banking • Work with Business Partners and Product Owners to translate business problems into appropriate Machine Learning use cases and plan the technical roadmap for short, mid , long term alignment • Formulate right hypothesis and implement algorithms to accept/reject the same • Be an expert in baselining ML outcomes and build upon them • Use statistical methods, machine learning techniques in the context of the data and with the clear intuition of the underlying mathematics and data structure • Formulate the right optimization target and implement algorithmic methods to achieve the same • Proficiently implement post deployment model monitoring techniques, drift detections and subsequent management • Expertly navigate bias and explainability aspects of the algorithms and implement solutions wherever required to meet corresponding needs • Expertly use Deep Learning frameworks like CNN, RNN, LSTM and Attention on basis of prudent evaluation of the need • Use large scale data processing frameworks such as Spark, AWS EMR for feature engineering and be proficient across various data both structured and un-structured. • Build ML models across Public and Private Cloud environments including container-based Kubernetes environments. • Develop end-to-end ML pipelines necessary to transform existing applications and business processes into true AI systems. • Build both batch and real-time model prediction pipelines with existing application and front-end integrations. • You will collaborate to develop large-scale data modeling experiments, evaluating against strong baselines, and extracting key statistical insights and/or cause and effect relations **Job Description** As a Software Engineer II, Java & Apache Spark within Consumer and Community Banking, in Card Account Management, you serve as a seasoned member of an agile team to design and deliver trusted market-leading technology products in a secure, stable, and scalable way. You are responsible for carrying out critical technology solutions across multiple technical areas within various business functions in support of the firm's business objectives. *See* https://in.linkedin.com/jobs/search?keywords=Spark&location=India&geoId=102713980&f_C=1068&currentJobId=3756690144&position=1&pageNum=0 (last accessed on November 11, 2023). *See* https://builtin.com/job/engineer/software-engineer-ii-java-apache-spark/1883999 (last accessed on November 9, 2023).

See also https://jpmc.fa.oraclecloud.com/hcmUI/CandidateExperience/en/sites/CX_1002/requisitions/preview/210445169/?keyword=spark&mode=location (last accessed on November 13, 2023); https://jpmc.fa.oraclecloud.com/hcmUI/CandidateExperience/en/sites/CX_1002/job/210442431/?keyword=spark&mode=location (last accessed on November 13, 2023).

Upon information and belief, Chase implements Spark. Upon information and belief, Chase's systems include "at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area[.]" Spark collects large volumes of data that cannot fit on a single node and distributes slices of the data across multiple nodes. Spark automatically partitions the data into Resilient Distributed Datasets that operate in parallel. The Resilient distributed datasets are partitioned across multiple nodes and processed across at least one memory area.

Resilient Distributed Datasets (RDDs)

Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: *parallelizing* an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

See https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167 (last accessed on November 11, 2023)

Partitioning

There are generally collections of various data items of massive volumes, that cannot fit into a single node and have to be partitioned across multiple nodes. Spark automatically does this partitioning of Resilient Distributed Datasets and distributes these partitions across different nodes. Key points related to these partitions are

In-Memory Computation

Spark uses in-memory computation as a way to speed up the total processing time. In the in-memory computation, the data is kept in RAM (random access memory) instead of the slower disk drives. This is very helpful as it reduces the cost of memory and allows for pattern detection, analyzes large data more efficiently. Main methods that accompany this are *cache()* and *persist()* methods.

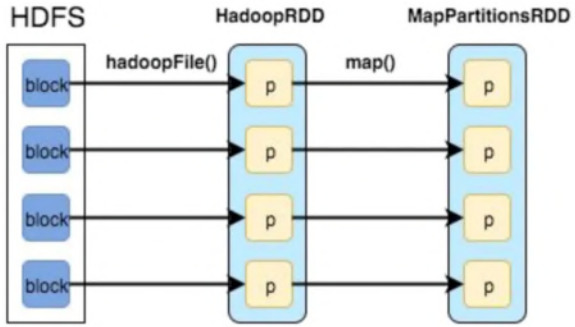
See https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 9, 2023).

The RDDs contain variables that allow the nodes to process slides of data in parallel. RDDs are processed using functions and multiple variables that can be used to create other RDDs to further process the data set.

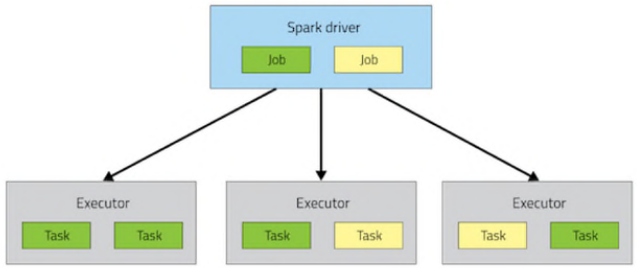
RDD Creation

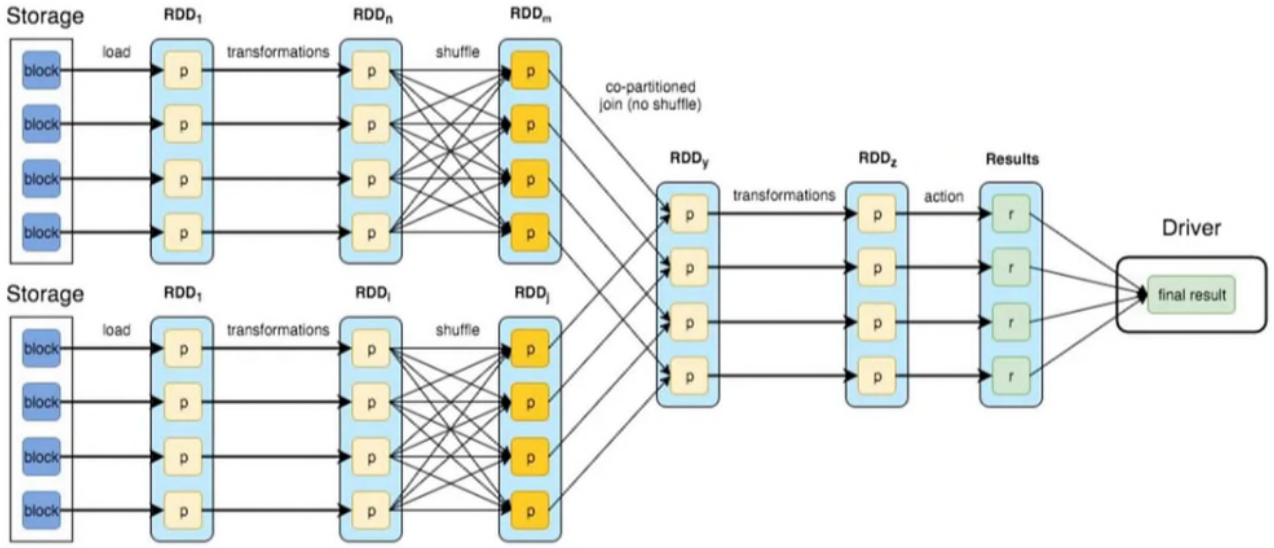
Here's an example of RDDs created during a method call:

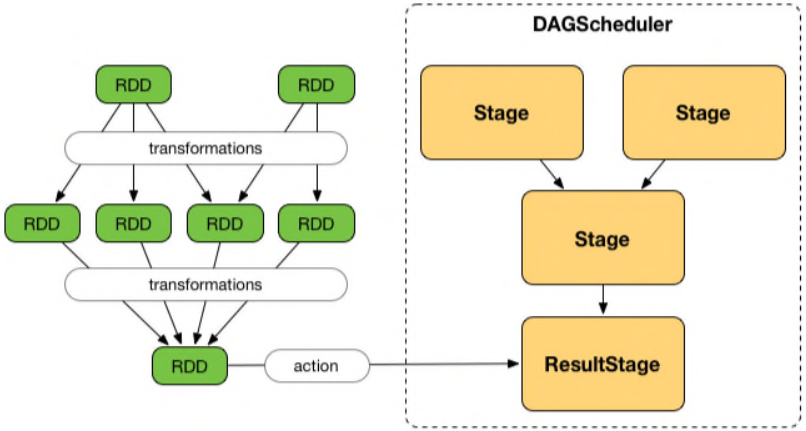
Which first loads HDFS blocks in memory and then applies map() function to filter out keys creating two RDDs:

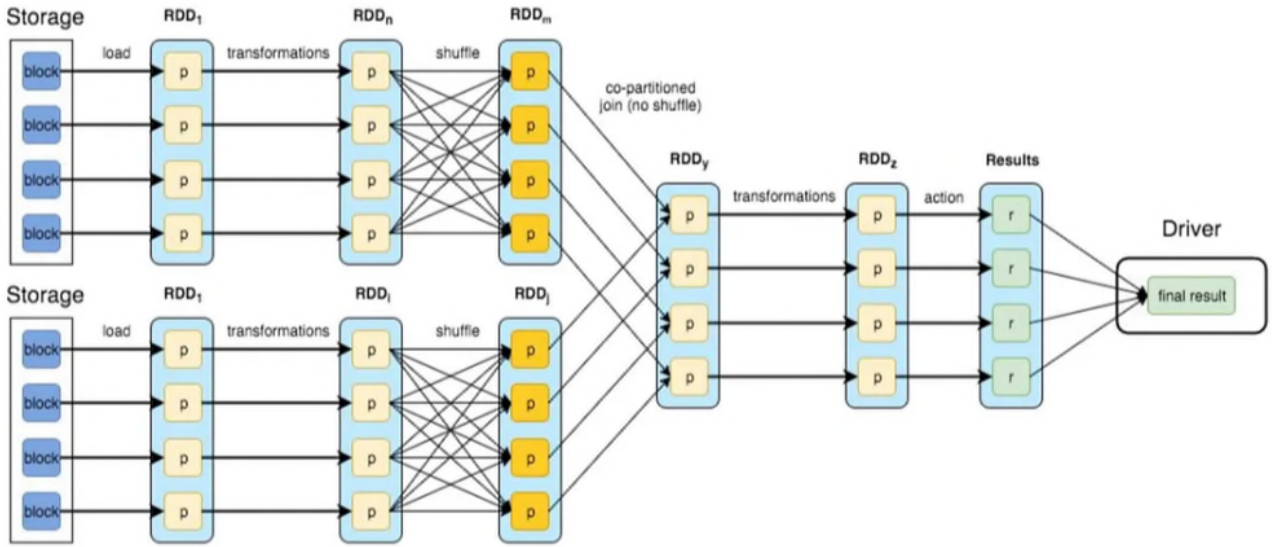


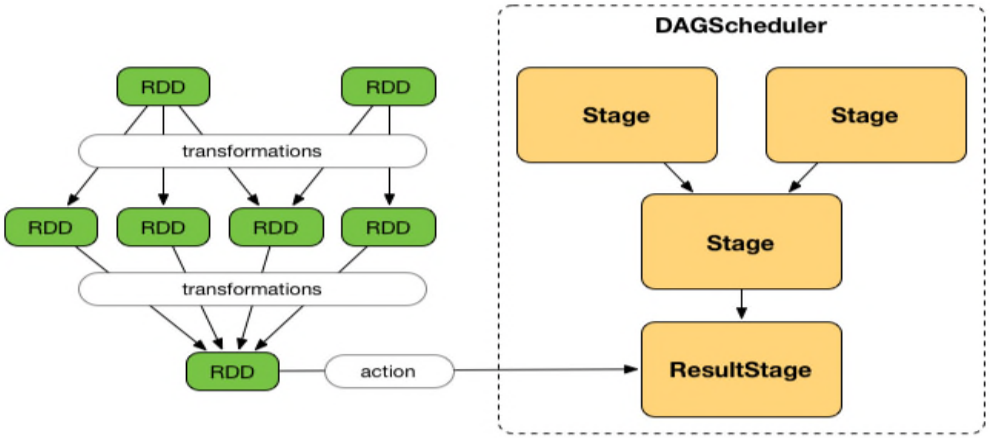
See https://spark.apache.org/docs/latest/rdd-programming-guide.html (last accessed on November 9, 2023).

	 Invoking an action inside a Spark application triggers the launch of a **job** to fulfill it. Spark examines the dataset on which that action depends and formulates an execution plan. The execution plan assembles the dataset transformations into stages. A **stage** is a collection of tasks that run the same code, each on a different subset of the data. *See* https://docs.cloudera.com/documentation/enterprise/5-6-x/topics/cdh_ig_spark_apps.html (last accessed on November 9, 2023).
9[b][i] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable,	Upon information and belief, Chase's systems include "at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable[.]" Spark contains a driver program that is configured to operate in the at least one processor and is configured to access the RDDs. Spark jobs are authored in a language such as Java, Scala, Phthon, or R. Spark is configured to access the at least one distributed shared variable (the RDDs) through the access of RDDs by a Spark job. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. *See* https://spark.apache.org/docs/1.2.1/programming-guide.html#:~:text=Go%20from%20Here-,Overview,parallel%20operations%20on%20a%20cluster. (last accesed November 9, 2023).

	Spark Overview Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. *See* https://spark.apache.org/docs/2.0.1/ (last accessed on November 9, 2023).
9[b][ii] and configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program	Upon information and belief, Chase's systems are "configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program[.]" The Spark Execution Model transforms the distributed sequential computing program into a series of stages and sets of parallel tasks that run in parallel on the Spark cluster. This includes Spark tasks operating on a set of RDDs in parallel in a given stage.  *See* https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167 (last accessed on November 9, 2023).

9[b][iii] when at least one intermediate condition occurs within the at least one distributed sequential program	Upon information and belief, Chase's systems include "at least one intermediate condition occurs within the at least one distributed sequential program[.]" Spark implements shuffle boundary where stages/tasks must wait for the previous stage to finish before they fetch map outputs. DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. *See* https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html (last accessed on November 9, 2023).

9[b][iv] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations,	Upon information and belief, Chase's systems are configured such that "wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations[.]" Spark implements a set of tasks running in parallel for a given stage. The distributed parallel computing program includes the Spark job and driver program (above). The spawned child distributed sequential computing program is the sequential code in a task for a given stage (above).  *See* https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167 (last accessed on November 9, 2023).

9[b][v] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.	Upon information and belief, Chase's systems are configured such that "wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation." Spark implements shuffle boundary where stages/tasks must wait for the previous stage to finish before they fetch map outputs. DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. *See* https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html (last accessed on November 10, 2023)